Improving The Performance of Virtual Labs Bubble Sort Experiment

Krutam Hathi
Software Engineering Research Center
International Institute of Information Technology
Hyderabad, India
krutam.hathi@researhc.iiit.ac.in

Raj Agnihotri
Virtual Labs
Hyderabad, India
raj@vlabs.ac.in

Venkatesh Choppella
Software Engineering Research Center
International Institute of Information Technology
Hyderabad, India
venkatesh.choppella@iiit.ac.in

2021 9th International Conference on Information and Education Technology (ICIET) | 978-1-6654-1933-8/21/$31.00 ©2021 IEEE | DOI: 10.1109/ICIET51873.2021.9419632

***Abstract*—Online education has witnessed a spike during the COVID-19 pandemic[1], due to which it is important to be performance efficient and serve the content in a reasonable time. In this case study of Virtual Labs, an e-learning platform we analyse and propose a set of approaches which helps in improving the website's performance. Virtual Labs is an e-learning website for virtual experiments in topics on science and engineering. These experiments are in the form of static webpages. This paper delineates the steps we followed to improve the performance of the static e-learning website. It shares insights into some of the concepts which can help improve the website performance.**

***Keywords*—e-learning, website performance, optimization, software engineering, education technology**

I. Introduction

*a) **About Virtual Labs.**:* Virtual Labs is an e-learning platform which is divided into units called **experiments**. Each experiment for example Data Structures, VLSI etc covers the concepts of respective topics taught in various engineering courses in India. A Virtual Labs experiment is a static website with content of type image, text, video or interactive javascript. A static website is a website which consists of pre-existing webpage files which gets loaded when requested by the browser[2].

*b) **Scope For Performance Improvement.**:* The performance of a website is one of the major factors which contribute to user experience. In this era of growing online education it is also important to keep a check on the performance of the e-learning websites. While studying the user experience metrics of Virtual Labs experiments, we came across various metrics which had scope for improvement which could indeed improve the user experience.

*c) **Problem Statement.**:* Virtual labs is being used by millions of people accross the country. Virtual Labs has more than 11,000 users per day accessing the website[3]. From the user's perspective, the webpages took more than 10 seconds to load for a fast 3G connection[4] and for a slow 3G connection it was as high as 20 seconds[5]. This led to the beginning of our work to improve the performance of the Virtual Labs experiments webpages.

*d) **Approach Brief.**:* This paper discusses optimization approaches for the different components of a static website like image, video, css etc in order to improve the performance of the website. It about the tool used to measure the performance of a website and tools which can optimise the website components. The research started off by listing all the variables that, according to us, might determine the performance of Virtual Labs website. We explored all the possible modifications that can be done to the variables and evaluated relative changes in the website performance for each modification. The improvement in the performance has proven helpful to millions of people who are using the Virtual Labs platform across the country.

*e) **Results Brief.**:* This research has helped us identify a set of approaches that can be used to improve the website's performance having js, html, css, images and video resources. Implementing these approaches has helped us bring down the webpage load time under 1.3 seconds from 6 seconds and the repeat view load time to under 0.3 seconds from 0.7 seconds. This shows the effectiveness of the approaches followed to improve the website performance.

In the next section we will discuss the approach we took to carry out this research.

II. Approach

*a) **Virtual Labs Architecture.**:* In this research we have worked on Virtual Labs Data Structures experiments. Virtual Labs Data Structure experiments are built using a custom framework. The framework requires the experiment author to provide the experiment content in a single html file i.e **exp.html**. This content can be converted to a Virtual Labs experiment using a single command provided by the framework. Our objective was to optimise this framework to improve the website performance. This optimisation will help in improving performance of all 23 Data Structures experiments built using this framework.

*b) **Initial Approach and Findings.**:* We initially searched for free online tools which could help us determine the performance of a website. Based on our findings we decided to use webpagetest tool[6]. This tool lets the user customise the location, network speed and browser for the performance testing. We have tested our website's performance on fast 3G network speed, server location Mumbai and Google Chrome browser. Google Analytics data report shows that more than 80% of the users open the website using google chrome. As Virtual Labs mainly targets Indian college

978-1-6654-1933-8/21/$31.00 ©2021 IEEE

students, we have selected Mumbai as the testing location. After setting up the tool we tested the performance of the bubble sort experiment. By analysing its performance report[7] we broke down the website components into dependent and independent variables. Dependent variables are those which depend on some other variable for example the performance of the server depends on the size and type of the server. Independent variables are those variables which do not depend on any other variable for example the size of image. The list of independent variables are:

- Server
  - Caching policy
  - Resource size/compression
- Client Side
  - Number of network calls
  - Browser caching policy
  - Speed of the internet
  - Resource load time
    -> HTML Load Time
    -> Javascript Load Time
    -> CSS Load Time
    -> Image Load Time

For each of the dependent variable i.e the metrics, it can further be broken into independent variables which can be modified to test the performance change. After breaking the website down into individual components we started optimising each component. The performance change for each modification to the component was tested by hosting the website and testing it using the webpagetest tool.

In the next section we discuss the performance metrics and their definitions.

III. Performance Metrics

The webpagetest tool reports two different aspects of website performance, which are:

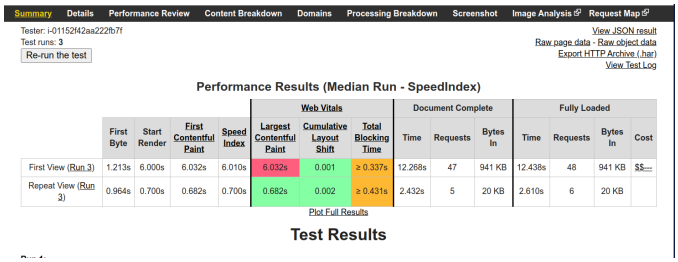
1) **First View:** The First View test is a test that was done with a browser that had its cache and cookies cleared out and represents what a first-time visitor to the page will experience.
2) **Repeat View:** The Repeat View test is a test that was done immediately after the First View test without clearing out anything.

For both of the above mentioned aspects, the result is divided into the below given metrics. These metrics are defined in the webpagetest documentation [8].

1) **Document Complete:** The metrics collected up until the browser considered the page loaded (onLoad event for those familiar with the javascript events). This happens after all of the images content have loaded but may not include content that is triggered by javascript execution. It is measured in seconds. This metric depends on other metrics like **No. Of Requests and Time** which are explained below in this section.
2) **Fully Loaded:** The metrics collected up until a 2 second pause in the network activity after Document Complete. It is measured in seconds. This metric depends on other metrics like **No. Of Requests and Time** which are explained below in this section.
3) **Requests:** This is the number of requests made by the browser for pieces of content on the page (images, javascript, css, etc). It is an integer with no associated unit.
4) **Start Render:** The first point in time when something was displayed on the screen. It is measured in seconds. For any webpage the html is rendered first after which other referrals, resources are requested over the network. Thus, in our case this is the time when the **exp.html** gets rendered which depends on the js and css libraries included before the <**body**> of the html.
5) **First Contentful Paint:** It marks the first point in the page load timeline where the user can see anything on the screen. The difference between FCP and start render is that start render ignores if the content is visible or invisible to the user. FCP is measured in seconds.
6) **Largest Contentful Paint:** It marks the point in the page load timeline when the page's main content has likely loaded. The unit for LCP is seconds. An LCP below 2.5 seconds is assumed to be good and more than 4 seconds is assumed to be poor[9].
7) **Speed Index:** It is the average time at which visible parts of the page are displayed. It is expressed in milliseconds and depends on the size of the viewport.

A. Performance Overview

By looking at the **summary** section of the report[10], the performance overview is divided into two parts:


Summary | Details | Performance Review | Content Breakdown | Domains | Processing Breakdown | Screenshot | Image Analysis | Request Map

Tester: i-0115 2f42aa222fb7f
Test runs: 3
Re-run the test

View JSON result
Raw page data - Raw object data
Export HTTP Archive (.har)
View Test Log

Performance Results (Median Run - SpeedIndex)

	First Byte	Start Render	First Contentful Paint	Speed Index	Largest Contentful Paint	Cumulative Layout Shift	Total Blocking Time	Document Complete Time	Document Complete Requests	Document Complete Bytes In	Fully Loaded Time	Fully Loaded Requests	Fully Loaded Bytes In	Cost
					Web Vitals									
First View (Run 3)	1.213s	6.000s	6.032s	6.010s	6.032s	0.001	≥ 0.337s	12.268s	47	941 KB	12.438s	48	941 KB	$$...
Repeat View (Run 3)	0.964s	0.700s	0.682s	0.700s	0.682s	0.002	≥ 0.431s	2.432s	5	20 KB	2.610s	6	20 KB	

Plot Full Results

Test Results

Fig. 1. Performance of Currently hosted Bubble Sort Experiment

- **First View**: The bubble sort experiment has a fully loading time of 12.438 seconds with number of requests made equal to 48 and the document complete time of 12.2 seconds with the number of requests equal to 47. It has a start render time of 6.0 seconds with the first byte coming at 1.213 seconds and a speed index of 6.1 seconds. The **largest contentful paint** time is 6.032 seconds. As the LCP for the webpage is greater than 4 seconds which means that the webpage has poor **largest contentful paint** and thus it needs to be optimised.
- **Repeat View**: The **fully loaded** time is 2.6 seconds with the number of request equal to 6 whereas the **document complete** time is 2.4 seconds with the number of requests

equal to 5. It has a **start render** time of 0.700 seconds with the **first byte** coming at 0.9 seconds and a **speed index** of 0.7 seconds.

The next section discusses about the procedure followed in order to improve the website's performance.

IV. Procedure

For each performance report analysis, we have broken down our approach into iterations and for each iteration the result was seen and thus concluded whether the change made was helpful in improving the performance of the website or not.

A. Iteration I: Parallel Loading Of Libraries

By looking at the **Details** section of the report[11] the first thing we noticed was that all the standard libraries along with other js and css resources were getting requested sequentially. According to the architecture all the required resources were being loaded at runtime i.e when the webpage is requested, by a javascript file. The green line in the **Details** section of the report shows the start render time. It can be seen that the sequential loading of resources increases the start render time and first contentful paint time.

Figure 2 shows sequential loading of the libraries and its effects on start render.

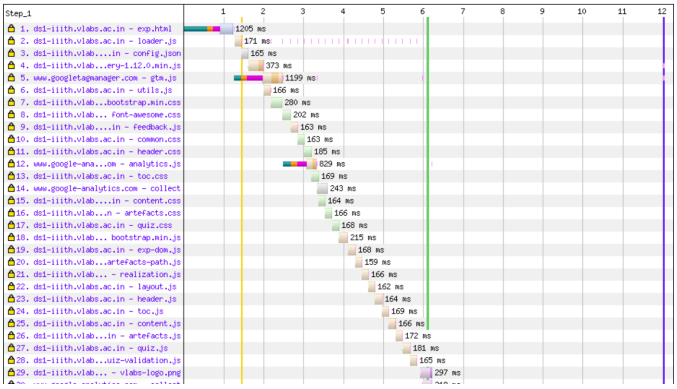


Fig. 2. Picture Showing Sequential Loading Of Libraries

To reduce this overhead we removed the involvement of the javascript file for loading the libraries at run-time. All the resources as well as the libraries were now loaded parallely by adding the resource path in html using **<link>** tag for css and **<script>** tag for javascript. Due to this modification, we are now able to leverage the parallel loading facility provided by the browser. The **Details** section of the report[12] shows that these changes helped in loading the javascript and css resources parallely and it reduced the start render time and first contentful paint time from 6 seconds to 5.5 seconds i.e by 9%.

B. Iteration II: Bundling And Minification Of Files

There were a total of 26 js files which were being loaded in different network calls. The total number of requests made to fully load a website was 48. To avoid these many network calls , we bundled and minified js files into one js file i.e main.js. Bundling is the process of combining the application code along with the code of all the libraries the application depends on[13].

There is now just a single network call made to load all the js resources. The result[14] of bundling all the js resources shows a reduction in the number of requests from 48 to 38 and also a reduction in start render time from 5.6 seconds to 4.6 seconds i.e by 18%.

This step was followed by minification of all the images present in the bubble sort experiment. For this reduction, we tried converting the image into different extensions like png, jpg, svg and webp. The results showed that though most of the images used in the experiment were size efficient but the best results in performance were seen when the images were converted to webp extension. So, all the images in the bubble sort experiment were converted to webp extension. One of the major changes were seen in the size of the Virtual Labs logo image. It's size reduced down from 29 Kb to 8Kb, which is a reduction of about 72.5% when converted to webp extension. Figure 4 shows the load time Virtual Labs logo initially.

The waterfall view under the **Details** section of the results show a reduction in the load time for the image from 297ms to 172ms i.e a reduction of 42%.

C. Iteration III: Using CDN for Standard Libraries

The experiment building framework uses some standard libraries like jquery, bootstrap, font-awesome. In this step we tried to use these cdn links for bootstrap,jquery and font-awesome instead of hosting it locally on the server.

The results show that the cdn took more time to load than local hosting. It can be observed from the report that when used cdn, bootstrap, jquery and font-awesome took more than 1 second to load whereas when hosted locally they took only 300-400 ms to load which is a reduction of 600ms i.e by 60%. Locally hosting these libraries also gave us an upper hand to customise the caching policy.

Figure 7 shows the reduction in load time of the libraries when hosted locally.

D. Iteration IV: Shifting Runtime Tasks At Build Time

The initial architecture of serving the web pages was such that, all the content of the website was present in a single html page, exp.html. Whenever a new webpage was requested, a javascript loader file was executed and the required webpage was built at runtime by modifying the contents of the current page. Thus for a new request made, the content in the current webpage had to be replaced with the new content and the same page is served. This made our website a single page application. Thus, everytime a new request is made, there needs to be some work done during the runtime before the webpage can be served.

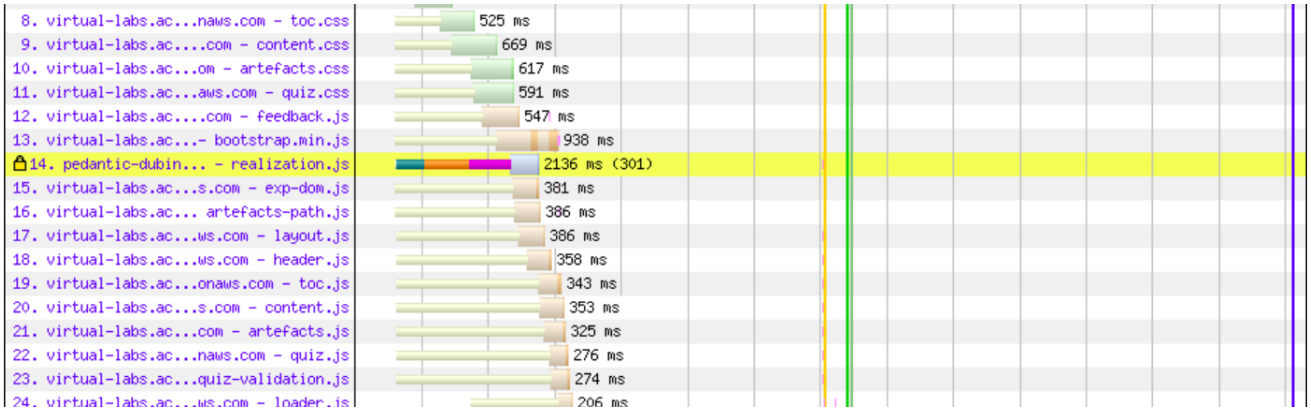


Fig. 3. Picture Showing Parallel Loading Of Libraries

Fig. 4. Load Time Of The Logo in png Extension

Fig. 5. Load Time Of The Logo In webp Extension

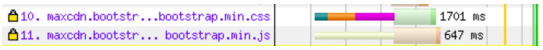


Fig. 6. Load Time of Libraries using CDN

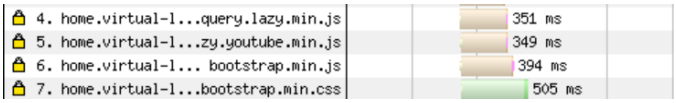
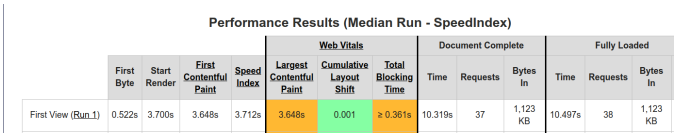


Fig. 7. Load Time of Libraries When Hosted Locally



Performance Results (Median Run - SpeedIndex)

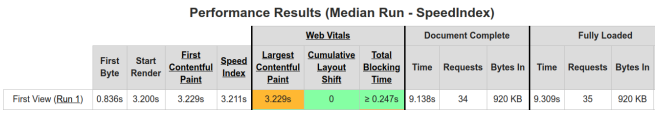
	First Byte	Start Render	First Contentful Paint	Speed Index	Web Vitals: Largest Contentful Paint	Web Vitals: Cumulative Layout Shift	Web Vitals: Total Blocking Time	Document Complete: Time	Document Complete: Requests	Document Complete: Bytes In	Fully Loaded: Time	Fully Loaded: Requests	Fully Loaded: Bytes In
First View (Run 1)	0.522s	3.700s	3.648s	3.712s	3.648s	0.001	≥ 0.361s	10.319s	37	1,123 KB	10.497s	38	1,123 KB

Fig. 8. Performance Before Shifting From Runtime To Build Time

This step was aimed at shifting the runtime tasks to build time by making web pages, in the form of static html, available directly from the server. The basic idea was to generate all the html files and then serve when requested without rebuilding it in every request. This change eliminated the following tasks during runtime:

- Work done browser for updating the webpage
- Involvement of the javascript loader file which had to be executed by to update the webpage

The results[15] of this step show that the number of requests for fully loaded webpage has been reduced from 38 to 35 and it has also reduced the start render time by 16% from 3.8 seconds to 3.2 seconds.



Performance Results (Median Run - SpeedIndex)

	First Byte	Start Render	First Contentful Paint	Speed Index	Web Vitals: Largest Contentful Paint	Web Vitals: Cumulative Layout Shift	Web Vitals: Total Blocking Time	Document Complete: Time	Document Complete: Requests	Document Complete: Bytes In	Fully Loaded: Time	Fully Loaded: Requests	Fully Loaded: Bytes In
First View (Run 1)	0.836s	3.200s	3.229s	3.211s	3.229s	0	≥ 0.247s	9.138s	34	920 KB	9.309s	35	920 KB

Fig. 9. Performance After Shifting From Runtime To Build Time

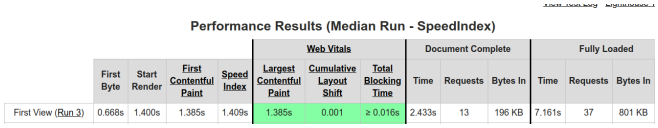
E. Iteration V: Improving Video Loading Time

After the previous step, on analysing the report it was found that the most time consuming process was loading the youtube video embedded in the webpage. Lazy loading can be helpful for saving this time. Lazy loading is a way to not load a resource until it is actually required[16]. So, by lazy loading a video, the video will not be loaded until it is clicked on.

In this step we analysed different video streaming sites like daily motion, vimeo and youtube to embed videos and ways to lazy load them.

The results show that there was not much difference in the websites performance in using any of these video streaming platforms and their lazy loading libraries. So, we decided to go with youtube for streaming videos due to the fact that it is more popular than the other two and they are constantly working to improve the video embedding feature.

The result of these changes shows that the first contentful paint has been reduced from 3.2 seconds to 1.4 seconds. It also shows the reduction in document complete requests to 13 and fully loaded requests has increased to 37 due to the fact that loading of the video is done after document is loaded.



Performance Results (Median Run - SpeedIndex)

	First Byte	Start Render	First Contentful Paint	Speed Index	Web Vitals: Largest Contentful Paint	Web Vitals: Cumulative Layout Shift	Web Vitals: Total Blocking Time	Document Complete: Time	Document Complete: Requests	Document Complete: Bytes In	Fully Loaded: Time	Fully Loaded: Requests	Fully Loaded: Bytes In
First View (Run 3)	0.668s	1.400s	1.385s	1.409s	1.385s	0.001	≥ 0.016s	2.433s	13	196 KB	7.161s	37	801 KB

Fig. 10. Performance Report After Lazy Loading Videos

F. Iteration VI: Applying Caching Policy

It was observed from the initial report[17] that total load time for repeat view is more than 3.5 seconds which can be optimised by using effective caching policy. So, in this step we analysed different caching policies and select one of them to be applied.

We are using server side caching[18] as well as service worker to cache the contents. On the server side, all the standard libraries like bootstrap, jquery and font-awesome have been cached with the expiration period of an year. Javascript and css resources are also being cached for a month and html resources are cached for a week. The reason behind the expiration period of the resources is that the javascript and css files are not expected to change frequently but the html resources are expected to change whenever the author of the experiment wants to change some topic in the experiment. Thus, the html caching is just for a week.

Pages which are heavy and have high chances of being accessed from an experiment are being pre-cached. Service worker is used to fetch and cache pages like simulations page of an experiment when any page of the experiment is accessed. Google analytics data helped us to find the highest page views for the experiment. A service worker does not hinder the website's performance because it is a script that the browser runs in the background, separate from a web page, allowing resources to be fetched without user interaction[19]. Service Worker also caches all the js, css and image resources for a week in the browser cache which helps to provide offline experience to the user. It means that once a page is opened in the browser the resources which are cached can be accessed without internet within the expiration period.

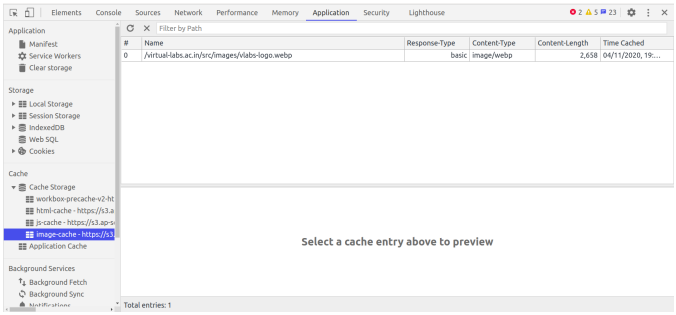


Fig. 11. Images cached by Service Worker

The result [20] shows that the repeat view first contentful paint time has been reduced by 85% from 2 seconds to 0.3 seconds. It is also evident from the report that the number of request has increased from 0 in document complete to 7 in fully loaded due to the reason that service worker makes request for the files to be pre-cached.

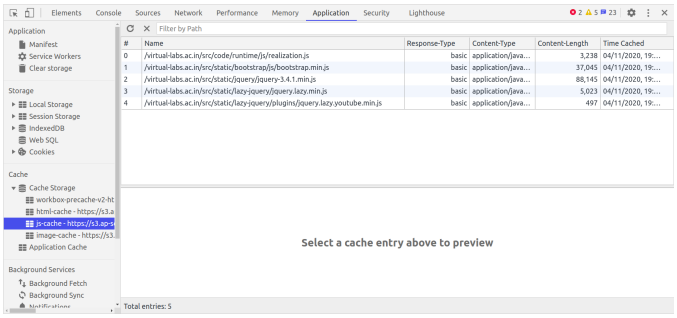


Fig. 12. JS cached by Service Worker

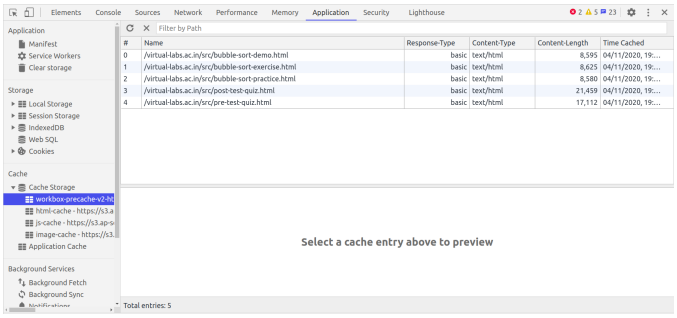
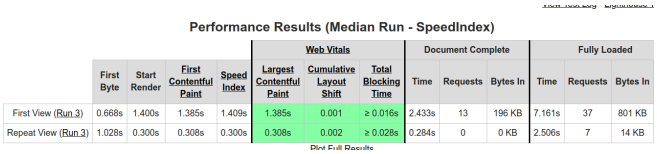


Fig. 13. Pre-caching By Service Worker



Performance Results (Median Run - SpeedIndex)

	First Byte	Start Render	First Contentful Paint	Speed Index	Web Vitals: Largest Contentful Paint	Web Vitals: Cumulative Layout Shift	Web Vitals: Total Blocking Time	Document Complete: Time	Document Complete: Requests	Document Complete: Bytes In	Fully Loaded: Time	Fully Loaded: Requests	Fully Loaded: Bytes In
First View (Run 3)	0.668s	1.400s	1.385s	1.409s	1.385s	0.001	≥ 0.016s	2.433s	13	196 KB	7.161s	37	801 KB
Repeat View (Run 3)	1.028s	0.300s	0.308s	0.300s	0.308s	0.002	≥ 0.028s	0.284s	0	0 KB	2.506s	7	14 KB

Plot Full Results

Fig. 14. Repeat view Performance After Applying Caching

V. Results

Performance Results (Median Run - SpeedIndex)

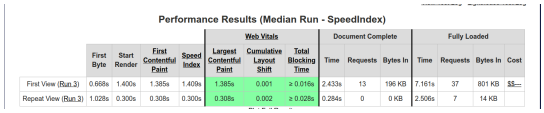
	First Byte	Start Render	First Contentful Paint	Speed Index	Web Vitals: Largest Contentful Paint	Web Vitals: Cumulative Layout Shift	Web Vitals: Total Blocking Time	Document Complete: Time	Document Complete: Requests	Document Complete: Bytes In	Fully Loaded: Time	Fully Loaded: Requests	Fully Loaded: Bytes In	Cost
First View (Run 3)	1.213s	6.000s	6.032s	6.010s	6.032s	0.001	≥ 0.337s	12.268s	47	941 KB	12.438s	48	941 KB	$$---
Repeat View (Run 3)	0.964s	0.700s	0.682s	0.700s	0.682s	0.002	≥ 0.431s	2.432s	5	20 KB	2.610s	6	20 KB	

Plot Full Results

Fig. 15. Performance of Website Before This Research

From the initial report[21] and the final report[22] the result can be divided into two parts, the first view and the repeat view.

Analysing the first view gives an idea about websites performance when the user opens it for the first time. Analysing the repeat view time and stats will let us know how effective the caching policy that we have implemented is.



Performance Results (Median Run - SpeedIndex)

	First Byte	Start Render	First Contentful Paint	Speed Index	Web Vitals: Largest Contentful Paint	Web Vitals: Cumulative Layout Shift	Web Vitals: Total Blocking Time	Document Complete: Time	Document Complete: Requests	Document Complete: Bytes In	Fully Loaded: Time	Fully Loaded: Requests	Fully Loaded: Bytes In	Cost
First View (Run 3)	0.668s	1.400s	1.385s	1.409s	1.385s	0.001	≥ 0.016s	2.433s	13	196 KB	7.161s	37	801 KB	$$---
Repeat View (Run 3)	1.028s	0.300s	0.308s	0.300s	0.308s	0.002	≥ 0.028s	0.284s	0	0 KB	2.506s	7	14 KB	

Fig. 16. Performance of Website After This Research

A. First View

By looking at the reports, it can be observed that the start render time and largest contentful paint has reduced from 6 seconds to 1.4 seconds where as the document complete load time has reduced from 12.4 seconds to 2.433 seconds. It is also evident that these set of approaches has helped in reducing the fully loaded time from 12.4 seconds to 7.1 seconds and also the total number of requests by 23% i.e from 48 to 37.

There is a major difference in document complete time and fully loaded time due to the fact that all the embedded youtube videos are being lazy loaded. Thus, after the document is loaded the videos start loading resulting in the difference in document complete time and fully loaded time.

The number of requests has reduced from 44 to 13 in document complete is due to the reduction of all the js file which was used at runtime to build the experiment. The number of requests is 37 for fully loaded webpage due to the reason that videos are being loaded after the document completes and the service worker makes some requests to pre-cache the required resources.

B. Repeat View

The repeat view performance has shown that the start render time has reduced by 58% from 0.7 seconds to 0.3 seconds by using this caching policy. The document complete time is also reduced from 2.4 seconds to 0.284 seconds and the document complete request has reduced from 5 to 0.

It can also be observed that the fully loaded request has increased from 6 to 7 due to the reason that we are using service worker to pre-cache certain resources after page load. This led to making new requests and thus increasing the number of requests.

VI. Conclusion

This research has helped us to understand the performance aspect of a website. We have discovered various open source tools for website performance testing. It has helped us test and analyse a websites performance and based on the analysis improve the performance of the website.

The insights from our analysis were put to test for improving the performance of a DS experiment, Bubble Sort. It has help us identify a set of approaches which when followed by the developers can be helpful in improving the performance of static websites which consist of resources like images, videos, javascript, css and html.

VII. Future Work

The results of this research has been encouraging and we see potential in exploring in further in some of the areas mentioned below:

- Apply the set of approaches to other DS experiments as well to improve and verify the approaches proposed.
- This paper suggests a set of guidelines which will be beneficial to make a framework based on these set of approaches when created as a library. This can help in creating an authoring framework for the authors to create experiments.
- Enhance the pre-caching policy using google analytics data. With more and more data getting collected over time, we can always enhance the pre-caching policy on the browser to further improve the performance.

References

[1] Litao Sun, Yongming Tang, and Wei Zuo. Coronavirus pushes education online. *Nature Materials*, 19 (6):687–687, Jun 2020. ISSN 1476-4660. doi: 10.1038/s41563-020-0678-8. URL https://doi.org/10.1038/s41563-020-0678-8.

[2] HAN Shuang-wang. The comparative research of dynamic website and static website. *Automation & Instrumentation*, (6):2, 2011.

[3] Virtual labs analytics summary - overall. https://datastudio.google.com/u/0/reporting/1bVjKkAw-e617LmNE1v_WPdIByVRz2waa/page/5fLPB.

[4] Webpagetest test result - mumbai : ds1-iii...0sort%20experiment - 11/11/20 20:21:30. https://www.webpagetest.org/result/201111_DiNJ_b821f4f9eacb372ed24c82007b52faa0/, .

[5] Webpagetest test result - mumbai : ds1-iii...0sort experiment - 11/11/20 20:09:27. https://www.webpagetest.org/result/201111_Di4J_961691f447e099319cad66a3c47a06a3/, .

[6] Webpagetest - website performance and optimization test. https://www.webpagetest.org/, .

[7] Webpagetest test result - mumbai : ds1-iii...bble-sort/exp.html - 10/27/20 10:52:22. https://www.webpagetest.org/result/201027_Di60_b40bf52e319090e2a0d2b39141c96fe5/, .

[8] Webpagetest test details - mumbai : virtual...0sort%20experiment - 10/30/20 16:21:18. https://www.webpagetest.org/result/201030_DiZW_ac3d841bfaeeae5803f001973e7f0e93/1/details/.

[9] Largest contentful paint (lcp). https://web.dev/lcp/.

[10] Webpagetest test result - mumbai : ds1-iii...bble-sort/exp.html - 10/27/20 10:52:22. https://www.webpagetest.org/result/201027_Di60_b40bf52e319090e2a0d2b39141c96fe5/, .

[11] Webpagetest test details - mumbai : ds1-iii...bble-sort/exp.html - 10/27/20 10:52:22. https://www.webpagetest.org/result/201027_Di60_b40bf52e319090e2a0d2b39141c96fe5/1/details/, .

[12] Webpagetest test result - mumbai : virtual...0sort%20experiment - 10/30/20 15:39:37. https://www.webpagetest.org/result/201030_DiAC_28bb226168a947eecfed067892b1ec10/, .

[13] Hernán Ceferino Vázquez, Alexandre Bergel, Santiago Vidal, JA Díaz Pace, and Claudia Marcos. Slimming javascript applications: An approach for removing unused functions from javascript libraries. *Information and Software Technology*, 107:18–29, 2019.

[14] Webpagetest test result - mumbai : virtual...0sort%20experiment - 10/30/20 16:00:34. https://www.webpagetest.org/result/201030_DiQ4_80c3a329f38deee810978b05b78b8584/, .
[15] Webpagetest test result - mumbai : s3.aps...%20experiment.html - 10/31/20 15:03:18. https://www.webpagetest.org/result/201031_DiTR_2159987d8fd0ee69521d0c5630273bcc/, .
[16] Debi Mishra, Sushil Baid, Ramesha Chandrasekhar, and Nikhil Jain. Lazy loading with code conversion, July 11 2006. US Patent 7,076,785.
[17] Webpagetest test result - mumbai : ds1-iii...bble-sort/exp.html - 10/27/20 10:52:22. https://www.webpagetest.org/result/201027_Di60_b40bf52e319090e2a0d2b39141c96fe5/, .
[18] Allen Yu. Client enhanced server-side cache system, September 26 2002. US Patent App. 09/816,994.
[19] Service workers: an introduction — web fundamentals. https://developers.google.com/web/fundamentals/primers/service-workers.
[20] Webpagetest test result - mumbai : home.vi...s.ac.in/index.html - 06/13/20 11:37:03. https://www.webpagetest.org/result/200613_B0_9affc58f275fd0dbf190b86d0be7cc3f/, .
[21] Webpagetest test details - mumbai : ds1-iii...bble-sort/exp.html - 10/27/20 10:52:22. https://www.webpagetest.org/result/201027_Di60_b40bf52e319090e2a0d2b39141c96fe5/1/details/, .
[22] Webpagetest test result - mumbai : home.vi...s.ac.in/index.html - 06/13/20 11:37:03. https://www.webpagetest.org/result/200613_B0_9affc58f275fd0dbf190b86d0be7cc3f/, .